HA.C

```
/*
 *              HA.C - Copyright (c) 1992 Echelon Corporation
 *
 * Last revision 5/10/92
 *
 * Example LONWORKS host application.  Uses a LONWORKS standard network
 * driver to communicate with an Echelon Serial LONTALK Adapter (SLTA)
 * or a Microprocessor Interface Program (MIP)-based network interface.
 *
 * This program is written in ANSI C so that it may be ported to any
 * host.  It has been tested on a PC using Borland C++ versions 2.0
 * and 3.0.  The application program is essentially the same as that
 * described in the Echelon engineering bulletin "NEURON CHIP-
 * based Installation of LONWORKS Networks" 005-0022-01 Rev B.
 * See page 22 of that document for a description of this application.
 *
 * The user interface is different, however.  See the sign-on
 * message for details.
 *
 * This program installs, updates, monitors, and watches SENSOR and ACTUATOR
 * nodes.  SENSOR nodes must have a one byte network variable output as the
 * first network variable, and a program ID of 'SENSOR'.  ACTUATOR nodes must
 * have a one byte network variable input as the first network variable, and
 * a program ID of 'ACTUATOR' or 'LAMP'.  The SENSOR.NC and LAMP.NC programs
 * included with the LONBUILDER software can be used with this program.  See
 * 'Running Your First Application' in Chapter 3 of the LONBUILDER Startup
 * and Hardware Guide for a description of the SENSOR and LAMP applications.
 *
 * The object code for this file must be linked with the object code
 * from NI_MSG.C.  The 'MAKEFILE' file can be used with Borland make
 * and the Borland C compiler to compile and link the HA.C and NI_MSG.C
 * programs.
 */

/****************************** Include Files ********************************/

#include "ni_msg.h"                 /* Network interface data structures    */
#include "ni_mgmt.h"                /* Network management data structures   */

#include <conio.h>                  /* For kbhit(), getch()                 */
#include <ctype.h>                  /* For toupper()                        */
#include <mem.h>                    /* For memcmp(), memset(), memcpy()     */
#include <stdio.h>                  /* For printf()                         */
#include <stdlib.h>                 /* For atoi(), exit()                   */
#include <string.h>                 /* For strlen(), strncpy()              */
```

```
/*
 ****************************************************************************
 * LON database.  This database is kept in RAM, and is therefore not
 * preserved between program invocations.  The database could be moved
 * to disk to make it non-volatile.
 ****************************************************************************
 */

/* Define two classes of nodes in database.                                 */

typedef enum {
    sensor,
    actuator,
    NUM_CLASSES
} node_type;

/* Database record for each node.                                           */

typedef struct {
    byte neuron_id[NEURON_ID_LEN];
    byte group_id;
} node_rec_type;

#define NUM_NODES  10              /* Number of nodes of each class         */
#define NUM_GROUPS 10              /* May be up to 255                      */

node_rec_type node_rec[NUM_CLASSES][NUM_NODES];
                                   /* Node ID 0 not used                    */
byte group_size[NUM_GROUPS];       /* Size of each group                    */
                                   /* Group ID 0 not used                   */
byte num_nodes[NUM_CLASSES];       /* Number of members in each class       */

/* Define a subnet for each class.  Any valid subnet number may be used.    */

const byte SENSOR_SUBNET   = 10;
const byte ACTUATOR_SUBNET = 11;

/*
 * Define a network variable selector.  Any valid selector may be used.
 * This selector is used for all updates, polls, and connections.
 * Typically, selectors would be maintained in a network variable
 * configuration table; to keep the example simple, this application uses
 * a single hard coded selector.
 */

const byte NV_SELECTOR_HI = 0x0B;
const byte NV_SELECTOR_LO = 0x0E;
```

```
/***************************** Local Variables ******************************/

node_type       node_class;
byte            subnet_id;
byte            node_id;
byte            group_being_watched = 0;
byte            group_being_formed = 0;
byte            current_group_size;
node_rec_type   *node_rec_ptr;
char            lon_device_name[80] = "LON1";

/***************************** Message Tags *********************************/

typedef byte msg_tag;

#define netmgr_domain_tag    1      /* Update net mgr's domain table        */
#define assign_node_tag      2      /* Update new node's domain table       */
#define assign_selector_tag  3      /* Assign NV selector                   */
#define upd_old_gp_tag       4      /* Inform old group                     */
#define join_group_tag       5      /* Add node to group                    */
#define upd_new_gp_tag       6      /* Inform new group                     */
#define nv_poll_msg_tag      7      /* Poll network variable                */
#define nv_update_msg_tag    8      /* Update network variable              */
#define bind_msg_tag         9      /* Bind to network variable connection  */
#define req_status_tag      10      /* Request status tag                   */

msg_tag wait_for_tag;               /* Waiting for ack/response on this tag */

/************************** Function Prototypes *****************************/

void scheduler(void);                    /* Main event loop                 */
void process_cmd(void);                  /* Process keyboard command        */
void start_group(void);                  /* Process Group keyboard command  */
void poll_sensor(void);                  /* Process Poll keyboard command   */
void update_actuator(void);              /* Process Update keyboard command */
void watch_NV(void);                     /* Process Watch keyboard command  */
void query_status(void);                 /* Process Status keyboard command */
void toggle_verbose(void);               /* Process Verbose keyboard command*/
void poll_received(void);                /* Process poll response message   */
void nv_received(void);                  /* Process NV update message       */
void query_received(void);               /* Process query status response msg*/
void service_pin_received(void);         /* Process service pin message     */
void update_old_group_complete(void);    /* Continue updating group         */
void assign_node_succeeds(void);         /* Continue updating group         */
void assign_selector_succeeds(void);     /* Continue updating group         */
```

```
void join_group_succeeds(void);          /* Continue updating group          */
void create_join_group_msg(msg_tag tag);/* Create update address table msg  */
void create_update_group_members_msg(byte group_id, msg_tag tag);
                                         /* Update group address             */
void create_assign_address_msg(msg_tag tag);
                                         /* Join domain                      */
void create_assign_selector_msg(msg_tag tag);
                                         /* Update NV config                 */
void create_nv_msg(msg_tag tag);         /* Create NV poll or update msg     */
void create_query_status_msg(msg_tag tag);
                                         /* Create query status message      */

/*
 ****************************************************************************
 * main().  Display sign-on message, open network driver, and call the
 * scheduler.
 ****************************************************************************
 */

int main(int argc, char *argv[])
{
    NI_Code ni_error = NI_OK;        /* Error code                           */

    char sign_on_message[] = {
      "Welcome to the LONWORKS Host Application Demonstration Program.\n\n"
      "This program polls, writes, and watches network variables,\n"
      "and connects SENSOR and ACTUATOR nodes together.\n\n"
      "Enter one of the following commands by typing the indicated letter:\n\n"
      "    E -- (E)xit this application and return to DOS.\n"
      "    G -- Start or terminate a network variable (G)roup connection.\n"
      "    P -- (P)oll a network variable on a sensor node.\n"
      "    S -- Report network interface (S)tatus.\n"
      "    U -- (U)pdate a network variable on an actuator node.\n"
      "    V -- Toggle (V)erbose mode (displays all incoming"
           " and outgoing messages).\n"
      "    W -- (W)atch a network variable group connection.\n"
    };

    char flag_message[] = {
      "\n\n"
      "To invoke the LONWORKS Host Application Demonstration Program, enter:"
      "\n\n"
      "    ha [-Ddevice] [-V]\n\n"
      "    -D selects the network device name, the default is 'LON1'.\n"
      "    -V invokes verbose mode.\n"
    };
```

```
/* Process command line arguments. */
while (--argc) {
    if ((argv[argc][0] == '/') || (argv[argc][0] == '-')) {
        switch (toupper(argv[argc][1])) {
        case 'D':
            /* Device name */
            if (argv[argc][2]) {
                /* Device name specified, copy it */
                strncpy(lon_device_name, &argv[argc][2],
                        sizeof(lon_device_name));
            } else {
                ni_error = NI_NO_DEVICE;
                ni_error_display("HA", ni_error);
                printf(flag_message);
                return(ni_error);
            }
            break;

        case 'V':
            /* Set verbose mode */
            verbose_flag = TRUE;
            break;

        default:
            /* Invalid flag */
            ni_error = NI_INVALID_FLAG;
            ni_error_display("HA", ni_error);
            printf(flag_message);
            return(ni_error);
        }
    } else {
        ni_error = NI_INVALID_SEPARATOR;
        ni_error_display("HA", ni_error);
        printf(flag_message);
        return(ni_error);
    }
}

/* Open and initialize the network interface. */
if ((ni_error = ni_init(lon_device_name)) != NI_OK)
    return(ni_error);

/* Have the network interface join the domain. */
subnet_id = 1;
node_id = NET_MGMT_NODE_ID;
create_assign_address_msg(netmgr_domain_tag);
```

```
    if ((ni_error = ni_msg_send()) != NI_OK) {
        ni_error_display("Network interface error on join domain", ni_error);
        return(ni_error);
    }
    ni_timer_set(&ni_wait_timer, niLOCAL_WAIT);
    wait_for_tag = netmgr_domain_tag;

    /* Display sign-on message. */
    printf(sign_on_message);

    /* Invoke the main event loop.  This function never returns. */
    scheduler();

    return(ni_error);
}

/*
 ****************************************************************************
 * scheduler().  Main event loop of application program.  Check for events
 * and invoke the appropriate function for each event.
 ****************************************************************************
 */

void scheduler(void)
{
    boolean show_prompt_flag = TRUE;
    msg_tag incoming_tag = 0;
    NI_Code ni_error = NI_OK;

    char    prompt_message[] =
      "\n(G)roup, (P)oll, (S)tatus, (U)pdate, (V)erbose, (W)atch, or (E)xit: ";

    /* Loop continuously until an exit command is detected in process_cmd().*/
    while (TRUE) {
        /* Loop until an input is received from the network interface */
        do {
            /* Display prompt if needed. */
            if (show_prompt_flag) {
                show_prompt_flag = FALSE;
                printf(prompt_message);
            }

            /* Check for message from network interface. */
            ni_error = ni_msg_receive();

            /* Check for keyboard input. */
            if (ni_error == NI_KEY_RECEIVED) {
                /* Process keyboard input. */
                process_cmd();
```

```
            if (!wait_for_tag)
                show_prompt_flag = TRUE;
        }

        /* Check for timeout. */
        if (ni_error == NI_TIMEOUT)
            /* No response from network. */
            printf("\nNo response from network.\n");

    } while (ni_error != NI_OK);

    /* Handle incoming message from network interface. */

    incoming_tag = msg_in.msg_hdr.exp.tag;

    if (incoming_tag && (incoming_tag == wait_for_tag)) {
        /* Incoming tag matches tag we're waiting for. */
        ni_timer_set(&ni_wait_timer, 0);
        wait_for_tag = 0;
    }

    if (msg_in.msg_hdr.exp.response) {
        /* Response message received */
        switch (incoming_tag) {
        case nv_poll_msg_tag:
            /* Response to network variable poll received */
            poll_received();
            show_prompt_flag = TRUE;
            break;

        case req_status_tag:
            /* Response to query status received */
            query_received();
            show_prompt_flag = TRUE;
            break;

        case bind_msg_tag:
            /* Successfully updated address table on network interface. */
            /* There will be no completion event for this message. */
            join_group_succeeds();
            break;

        default:
            /* Ignore all other responses. */
            continue;
        }
        continue;
    }
```


Subnet 10 node 1

```
if (msg_in.msg_hdr.exp.cmpl_code == MSG_NOT_COMPL) {
    /* Incoming message received */
    if (msg_in.data.exp.code & 0x80)
        /* Handle network variable update */
        nv_received();
    else if (msg_in.data.exp.code == NM_service_pin) {
        /* Handle service pin message */
        service_pin_received();
        show_prompt_flag = TRUE;
    }

    continue;                /* Ignore all other incoming messages   */
}

/* Handle completion codes -- successes or failures. */

if (incoming_tag == upd_old_gp_tag) {
    /* All members of old group have been updated. */
    /* Ignore if failure occurred. */
    update_old_group_complete();
    continue;
} else if (incoming_tag == upd_new_gp_tag)
    /* All members of new group have been updated. */
    /* Ignore if failure occurred. */
    continue;

if (msg_in.msg_hdr.exp.cmpl_code == MSG_FAILS) {
    /* Ignore all other failure completion events. */
    printf("\nMessage failed, tag = %d.\n", incoming_tag);
    show_prompt_flag = TRUE;
    continue;
}

switch (incoming_tag) {
case assign_node_tag:
    /* Successfully updated domain table. */
    assign_node_succeeds();
    break;

case assign_selector_tag:
    /* Successfully assigned network variable selector. */
    assign_selector_succeeds();
    break;

case nv_update_msg_tag:
    /* NV update complete. */
    show_prompt_flag = TRUE;
    break;
```

```
        case join_group_tag:
            /* Successfully updated address table. */
            join_group_succeeds();
        }
    }
}

/*
 ****************************************************************************
 * process_cmd().  Process a command from the keyboard.
 ****************************************************************************
 */

void process_cmd(void)
{
    char ch;

    ch = getch();

    switch (toupper(ch)) {
        case CTRL_C:
        case 'E':
        case 'Q':
            /* Exit command. */
            printf("Exit.\n");
            printf("Leaving the host application demonstration program.\n");
            exit(NI_OK);

        case 'G':
            /* Group command. */
            printf("Group.\n");
            start_group();
            break;

        case 'P':
            /* Poll command. */
            printf("Poll.\n");
            poll_sensor();
            break;

        case 'S':
            /* Network interface status command. */
            printf("Status.\n");
            query_status();
            break;
```

```
        case 'U':
            /* Update network variable command. */
            printf("Update.\n");
            update_actuator();
            break;

        case 'V':
            /* Toggle verbose mode. */
            printf("Verbose.\n");
            toggle_verbose();
            break;

        case 'W':
            /* Watch network variable command. */
            printf("Watch.\n");
            watch_NV();
            break;

        case '\r':
            break;

        default:
            printf("\nDidn't recognize that command, enter E, G, P, S, U, V,"
                   " or W.\n");
    }
}

/*
 ****************************************************************************
 * start_group().  Process the Group keyboard command.  Toggle install
 * mode: either start a new group or terminate the current group definition.
 * Called from process_cmd().
 ****************************************************************************
 */

void start_group(void)
{
    if (group_being_formed) {
        /* Terminate this group. */
        printf("Group %d complete with %d members\n", group_being_formed,
            group_size[group_being_formed]);
        group_being_formed = 0;      /* Finished with this group             */
    } else {
        /* Start a new group.  Look for an unused group ID for the new group*/
        for (group_being_formed = 1; group_being_formed < NUM_GROUPS;
             group_being_formed++) {
```

```
            if (!group_size[group_being_formed]) {
                /* Found an unused group ID. */
                printf("Start forming group %d.\n", group_being_formed);
                printf("Press the service pin on nodes to add to the group.\n");
                printf("Press G again to finish this group.\n\n");
                return;
            }
        }
        /* No unused groups. Ignore the start group command. */
        printf("Too many groups\n");
        group_being_formed = 0;
    }
}

/*
 ****************************************************************************
 * poll_sensor().  Process the Poll keyboard command.  Read a node ID from
 * the keyboard and poll it.  Called from process_cmd().
 ****************************************************************************
 */

void poll_sensor(void)
{
    char input_buffer[80];

    subnet_id = SENSOR_SUBNET;
    printf("Enter node ID of sensor node to poll: ");
    ni_gets(input_buffer, sizeof(input_buffer));
    node_id = (byte) atoi(input_buffer);
    create_nv_msg(nv_poll_msg_tag);
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = nv_poll_msg_tag;
    }
}

/*
 ****************************************************************************
 * update_actuator().  Process the Update keyboard command.  Read a node ID
 * from the keyboard and update a network variable on the specified node.
 * Called from process_cmd().
 ****************************************************************************
 */

void update_actuator(void)
{
    byte data;
    char input_buffer[80];
```

```
    subnet_id = ACTUATOR_SUBNET;
    printf("Enter node id of actuator node to update: ");
    ni_gets(input_buffer, sizeof(input_buffer));
    node_id = (byte) atoi(input_buffer);
    printf("Enter data to send to actuator's network variable: ");
    ni_gets(input_buffer, sizeof(input_buffer));
    data = (byte) atoi(input_buffer);

    create_nv_msg(nv_update_msg_tag);
    msg_out.msg_hdr.exp.st = ACKD;    /* Change service type to acknowledged  */
    msg_out.data.unv.data[0] = data; /* Single byte data only                 */
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = nv_update_msg_tag;
    }
}

/*
 ****************************************************************************
 * watch_NV().  Bind an input network variable to a connection and monitor
 * any updates to the network variable.  To simplify the example, a single
 * network variable selector is used, and the first address table entry on
 * the node is always used.  Called from process_cmd().
 ****************************************************************************
 */

void watch_NV(void)
{
    int  group_to_watch;
    byte old_group_id;
    char input_buffer[80];

    old_group_id = group_being_watched;
    printf("Enter group ID of connection to watch (0 for none): ");
    ni_gets(input_buffer, sizeof(input_buffer));
    group_to_watch = (byte) atoi(input_buffer);

    if (old_group_id && (group_to_watch == old_group_id))
        /* This group ID is already being watched. */
        printf("Already watching group %d.\n", group_being_watched);
    else {
        group_being_formed = group_being_watched = (byte) group_to_watch;
        subnet_id = 1;              /* Subnet ID for this node              */
        node_id = NET_MGMT_NODE_ID; /* Node ID for this node                */
        /* Remove node from previous watch group, if any. */
```

```
        if ((old_group_id)
            && (--group_size[old_group_id] != 0)) {
            /* Update all remaining group members. */
            create_update_group_members_msg(old_group_id, upd_old_gp_tag);
            wait_for_tag = upd_old_gp_tag;
            if (verbose_flag)
                printf("\nChange the group size for group %d to %d members.\n",
                        old_group_id, group_size[old_group_id]);
        } else {
            /* No members left in old group, add node to new group. */
            create_join_group_msg(bind_msg_tag);
            wait_for_tag = bind_msg_tag;
            if (verbose_flag)
                printf("\nAdd-node %d to group %d.\n",
                        node_id, group_being_formed);
        }
        /* Send out next message. */
        if (ni_msg_send() == NI_OK)
            ni_timer_set(&ni_wait_timer, niNET_WAIT);
        else
            wait_for_tag = 0;
    }
}

/*
 ************************************************************************
 * query_status().  Request network interface status.  Called from
 * process_cmd().
 ************************************************************************
 */

void query_status(void)
{
    subnet_id = 1;
    node_id = NET_MGMT_NODE_ID;
    create_query_status_msg(req_status_tag);
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niLOCAL_WAIT);
        wait_for_tag = req_status_tag;
    }
}

/*
 ************************************************************************
 * toggle_verbose().  Toggle verbose mode.  When verbose mode is on, all
 * incoming and outgoing LONTALK messages are displayed.  Called from
 * process_cmd().
 ************************************************************************
 */
```

```
void toggle_verbose(void)
{
    verbose_flag = !verbose_flag;
    printf("Verbose ");
    if (verbose_flag)
        printf("on");
    else
        printf("off");
    printf(".  Incoming and outgoing LONTALK messages will ");
    if (!verbose_flag)
        printf("not ");
    printf("be displayed.\n");
}

/*
 ****************************************************************************
 * poll_received().  Process network variable poll response message.  Display
 * the first byte of the response.  Called from scheduler().
 ****************************************************************************
 */

void poll_received(void)
{
    printf("Data received from sensor node %d is %d\n",
            msg_in.addr.rcv.source.node, msg_in.data.unv.data[0]);
}

/*
 ****************************************************************************
 * nv_received().  Process network variable update message.  Display
 * the first byte of the update.  A more complete application would
 * maintain a network variable configuration table, and would determine
 * which network variable had been received by looking up the network
 * variable selector in the network variable configuration table.
 * This application uses a single hard-coded network variable selector
 * and does not maintain a network variable configuration table.
 * Called from scheduler().
 ****************************************************************************
 */
```

```
void nv_received(void)
{
    if (group_being_watched
     && (msg_in.addr.rcv.format == ADDR_RCV_GROUP)
     && (msg_in.addr.rcv.dest.gp == group_being_watched)
     && (msg_in.data.unv.NV_selector_hi == NV_SELECTOR_HI)
     && (msg_in.data.unv.NV_selector_lo == NV_SELECTOR_LO))
        printf("\nData received from node %d to group %d is %d\n",
                msg_in.addr.rcv.source.node, msg_in.addr.rcv.dest.gp,
                msg_in.data.unv.data[0]);
}

/*
 ******************************************************************************
 * query_received().  Process query status response message.  Display
 * the status.  Called from scheduler().
 ******************************************************************************
 */

void query_received(void)
{
    ND_query_status_response *status_ptr;

    status_ptr = (ND_query_status_response *) &msg_in.data.exp;
    printf("\nNetwork interface status:\n");
    printf("    Transmission (CRC) errors = %d\n", status_ptr->xmit_errors);
    printf("    Transaction timeouts = %d\n",
                status_ptr->transaction_timeouts);
    printf("    Receive transaction full errors = %d\n",
                status_ptr->rcv_transaction_full);
    printf("    Lost messages = %d\n", status_ptr->lost_msgs);
    printf("    Missed messages = %d\n", status_ptr->missed_msgs);
    printf("    Last reset cause = %d\n", status_ptr->reset_cause);
    printf("    Node state = ");
    switch ((nm_node_state) status_ptr->node_state) {
    case APPL_UNCNFG:
        printf("Application loaded, network image unconfigured\n");
        break;

    case NO_APPL_UNCNFG:
        printf("Application not loaded, network image unconfigured\n");
        break;

    case CNFG_ONLINE:
        printf("Application loaded and online, network image configured\n");
        break;
```

```
    case CNFG_OFFLINE:
        printf("Application loaded and hard offline,"
                " network image configured\n");
        break;

    case SOFT_OFFLINE:
        printf("Application loaded and soft offline,"
                " network image configured\n");
        break;

    default:
        printf("Unrecognized node state\n");
    }
    printf("    NEURON CHIP firmware version number = %d\n",
            status_ptr->version_number);
    printf("    NEURON CHIP model number = %d\n", status_ptr->model_number);
    printf("    Last error logged = %d\n", status_ptr->error_log);
}

/*
 ****************************************************************************
 * service_pin_received().  Process service pin message.  Add node to
 * current group if group installation is on.  Called from scheduler().
 ****************************************************************************
 */

void service_pin_received(void)
{
    byte num_nodes_in_class;
    byte old_group_id;
    NM_service_pin_msg *svc_pin_msg_ptr;

    if (!group_being_formed)
        /* Ignore service pin message if group installation is not on. */
        return;

    svc_pin_msg_ptr = (NM_service_pin_msg *) &msg_in.data.exp;
    printf("\nReceived a service pin message from ");

    if (!memcmp(svc_pin_msg_ptr->id_string, "SENSOR\x0\x0", ID_STR_LEN)) {
        /* Service pin message came from a sensor node. */
        subnet_id = SENSOR_SUBNET;
        node_class = sensor;
        printf("a sensor node\n");
    } else if ((!memcmp(svc_pin_msg_ptr->id_string, "ACTUATOR", ID_STR_LEN))
            || (!memcmp(svc_pin_msg_ptr->id_string, "LAMP\x0\x0\x0\x0",
                ID_STR_LEN))) {
```

```
    /* Service pin message came from an actuator node. */
    subnet_id = ACTUATOR_SUBNET;
    node_class = actuator;
    printf("an actuator node\n");
} else {
    /* Service pin message received from unrecognized node type. */
    printf("an unrecognized node\n");
    return;
}

/* Find this node in the database. Look through database for this class. */
num_nodes_in_class = num_nodes[node_class];

for (node_id = 1; node_id <= num_nodes_in_class; node_id++) {
    node_rec_ptr = &node_rec[node_class][node_id];
    if (!memcmp(svc_pin_msg_ptr->neuron_id, node_rec_ptr->neuron_id,
          NEURON_ID_LEN)) {
        /* NEURON ID matches database. */
        printf("This node is already in the LON database - "
                "subnet %d, node %d\n", subnet_id, node_id );

        /* Get the group ID that the node was in. */
        old_group_id = node_rec_ptr->group_id;

        if (old_group_id == group_being_formed)
            /* Attempt to add node to group it's already in. */
            return;

        printf("Removing this node from group %d\n", old_group_id);

        /* Mark database record as no longer in group. */
        node_rec_ptr->group_id = 0;

        /* Decrement size of group by one member. */
        if (--group_size[old_group_id] != 0) {
            /* Update all remaining group members. */
            create_update_group_members_msg(old_group_id, upd_old_gp_tag);
            wait_for_tag = upd_old_gp_tag;
            if (verbose_flag)
                printf("\nChange the group size for group %d to %d "
                    "members.\n", old_group_id, group_size[old_group_id]);
        } else {
            /* No members left, add node to new group. */
            create_join_group_msg(join_group_tag);
            wait_for_tag = join_group_tag;
            if (verbose_flag)
                printf("\nAdd node %d to group %d.\n",
                        node_id, group_being_formed);
        }
```

```
            /* Send out next message. */
            if (ni_msg_send() == NI_OK)
                ni_timer_set(&ni_wait_timer, niNET_WAIT);
            else
                wait_for_tag = 0;
            return;                  /* Known node, all done here          */
        }
    }

    /* Not a known node, add to the database. */

    if (num_nodes_in_class >= NUM_NODES - 1) {
        /* Database full--print error message and return. */
        printf("LON Database is full - can't add this node\n");
        return;
    }

    node_id = num_nodes_in_class + 1; /* Allocate next node id             */
    printf("Adding node to LON database - subnet %d, node %d\n",
            subnet_id, node_id);

    /* Save NEURON ID in database. */
    node_rec_ptr = &node_rec[node_class][node_id];
    memcpy(node_rec_ptr->neuron_id, svc_pin_msg_ptr->neuron_id,
           NEURON_ID_LEN);

    node_rec_ptr->group_id = 0;       /* Not yet in the new group          */
    create_assign_address_msg(assign_node_tag);
                                      /* Assign domain table entry         */
    if (verbose_flag)
        printf("\nAssign domain table entry.\n");
    /* Configure a new node. */
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = assign_node_tag;
    }
}

/*
 ****************************************************************************
 * update_old_group_complete().  Continue adding a node to a group.  Have
 * this node join the new group.  This function is called by scheduler()
 * when members of the old group have been updated.
 ****************************************************************************
 */
```

```
void update_old_group_complete(void)
{
    create_join_group_msg(join_group_tag);
    if (verbose_flag)
        printf("\nAdd node %d to group %d.\n", node_id, group_being_formed);
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = join_group_tag;
    }
}

/*
 ****************************************************************************
 * assign_node_succeeds().  Continue adding a node to a group.  Update
 * LON database with the new node's data and fill in NV config table.
 * This function is called by scheduler() when a new node has been assigned
 * its subnet and node IDs.
 ****************************************************************************
 */

void assign_node_succeeds(void)
{
    if (!group_being_formed)
        /* Return if a connection is not being created or updated. */
        return;

    /* Update LON database. Increment count in this class. */
    num_nodes[node_class] += 1;

    /* Update NV config table. */
    create_assign_selector_msg(assign_selector_tag);
    if (verbose_flag)
        printf("\nAssign network variable selector for node %d.\n", node_id);
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = assign_selector_tag;
    }
}

/*
 ****************************************************************************
 * assign_selector_succeeds().  Continue adding a node to a group.  Have the
 * node join the new group.  This function is called by scheduler() when the
 * network variable selector update has been completed.
 ****************************************************************************
 */
```

```
void assign_selector_succeeds(void)
{
    if (!group_being_formed)
        /* Return if a connection is not being created or updated. */
        return;

    create_join_group_msg(join_group_tag);
    if (verbose_flag)
        printf("\nAdd node %d to group %d.\n", node_id, group_being_formed);
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = join_group_tag;
    }
}

/*
 ************************************************************************
 * join_group_succeeds().  Continue adding a node to a group.  Update the
 * node record in the database and update the group members.  This function
 * is called by scheduler() when the node has joined the new group.
 ************************************************************************
 */

void join_group_succeeds(void)
{
    /* Update LON database. */
    group_size[group_being_formed] = current_group_size;
    node_rec_ptr->group_id = group_being_formed;

    /* Update all group members. */
    create_update_group_members_msg(group_being_formed, upd_new_gp_tag);
    if (verbose_flag)
        printf("\nChange the group size for group %d to %d members.\n",
                group_being_formed, current_group_size);
    if (ni_msg_send() == NI_OK) {
        ni_timer_set(&ni_wait_timer, niNET_WAIT);
        wait_for_tag = upd_new_gp_tag;
    }
}

/*
 ************************************************************************
 * create_join_group_msg().  Create the network management message to make
 * the current node join the current group.  The message is an update address
 * table entry network management message.
 ************************************************************************
 */
```

```
void create_join_group_msg(msg_tag tag)
{
    NM_update_addr_request *update_addr_msg_ptr;

    /* Get group size. */
    current_group_size = group_size[group_being_formed];
    if (current_group_size >= 63)
        /* Group is full, return. */
        return;

    /* Initialize network interface message buffer header and message tag. */
    ni_msg_hdr_init(sizeof(NM_update_addr_request));
    if (node_id == NET_MGMT_NODE_ID)
        msg_out.ni_hdr.q.q_cmd = niNETMGMT;
    msg_out.msg_hdr.exp.tag = tag;

    /* Create destination address (subnet/node format). */
    msg_out.addr.snd.sn.type      = SUBNET_NODE;
    msg_out.addr.snd.sn.domain    = 0;
    msg_out.addr.snd.sn.node      = node_id;
    msg_out.addr.snd.sn.rpt_timer = RPT_TIMER_AT;
    msg_out.addr.snd.sn.retry     = RETRY_AT;
    msg_out.addr.snd.sn.tx_timer  = TX_TIMER_AT;
    msg_out.addr.snd.sn.subnet    = subnet_id;

    /* Create a message to update an address table entry. */
    update_addr_msg_ptr = (NM_update_addr_request *) &msg_out.data.exp;
    update_addr_msg_ptr->code                = NM_update_addr;
    update_addr_msg_ptr->addr_index          = 0;
                                /* First address table entry          */
    if (group_being_formed) {
        /* Create group address table entry. */
        update_addr_msg_ptr->address.gp.type      = 1;
        update_addr_msg_ptr->address.gp.member    = current_group_size;
        update_addr_msg_ptr->address.gp.size      = ++current_group_size;
                                    /* One more in group              */
        update_addr_msg_ptr->address.gp.domain    = 0;
        update_addr_msg_ptr->address.gp.rpt_timer = RPT_TIMER_AP;
        update_addr_msg_ptr->address.gp.retry     = RETRY_AP;
        update_addr_msg_ptr->address.gp.tx_timer  = TX_TIMER_AP;
        update_addr_msg_ptr->address.gp.group     = group_being_formed;
    } else
        /* Unassign address table entry. */
        update_addr_msg_ptr->address.ua.type      = UNASSIGNED;
}
```

```
/*
 ****************************************************************************
 * create_update_group_members_msg().  Create the network management message
 * to update all members of the specified group with the new member count.
 * The message is an update group address table entry network management
 * message.
 ****************************************************************************
 */

void create_update_group_members_msg(byte group_id, msg_tag tag)
{
    NM_update_group_addr_request *update_group_msg_ptr;

    /* Initialize network interface message buffer header and message tag. */
    ni_msg_hdr_init(sizeof(NM_update_group_addr_request));
    msg_out.msg_hdr.exp.tag = tag;

    /* Create destination address (multicast format). */
    msg_out.addr.snd.gp.type      = 1;
    msg_out.addr.snd.gp.size      = group_size[group_id] + 1;
                    /* Add one for the network management tool (this node) */
    msg_out.addr.snd.gp.domain    = 0;
    msg_out.addr.snd.gp.member    = 0;      /* All members                  */
    msg_out.addr.snd.gp.rpt_timer = RPT_TIMER_AT;
    msg_out.addr.snd.gp.retry     = RETRY_AT;
    msg_out.addr.snd.gp.tx_timer  = TX_TIMER_AT;
    msg_out.addr.snd.gp.group     = group_id;

    /* Create the update a group address table entry message. */
    update_group_msg_ptr = (NM_update_group_addr_request *) &msg_out.data.exp;
                                  /* Get pointer to data part of message  */

    update_group_msg_ptr->code                  = NM_update_group_addr;
    update_group_msg_ptr->group_addr.type       = 1;
    update_group_msg_ptr->group_addr.size       = group_size[group_id];
                                 /* Domain and member are not used       */
    update_group_msg_ptr->group_addr.rpt_timer = RPT_TIMER_AP;
    update_group_msg_ptr->group_addr.retry      = RETRY_AP;
    update_group_msg_ptr->group_addr.tx_timer   = TX_TIMER_AP;
    update_group_msg_ptr->group_addr.group      = group_id;
}

/*
 ****************************************************************************
 * create_assign_address_msg().  Create the network management message to
 * assign a subnet ID and node ID to a new node.  The message is an update
 * domain table entry network management message.
 ****************************************************************************
 */
```

```
void create_assign_address_msg(msg_tag tag)
{
    NM_update_domain_request *update_domain_msg_ptr;

    /* Initialize network interface message buffer header and message tag. */
    ni_msg_hdr_init(sizeof(NM_update_domain_request));
    if (node_id == NET_MGMT_NODE_ID)
        msg_out.ni_hdr.q.q_cmd = niNETMGMT;
    msg_out.msg_hdr.exp.tag = tag;

    /* Create destination address (NEURON ID format). */
    msg_out.addr.snd.id.type       = NEURON_ID;
    msg_out.addr.snd.id.domain     = 0;
    msg_out.addr.snd.id.rpt_timer = RPT_TIMER_DT;
    msg_out.addr.snd.id.retry      = RETRY_DT;
    msg_out.addr.snd.id.tx_timer   = TX_TIMER_DT;
    msg_out.addr.snd.id.subnet     = subnet_id;
    memcpy(msg_out.addr.snd.id.nid, node_rec_ptr->neuron_id, NEURON_ID_LEN);

    /* Create the update a domain table entry message. */
    update_domain_msg_ptr = (NM_update_domain_request *) &msg_out.data.exp;

    update_domain_msg_ptr->code         = NM_update_domain;
    update_domain_msg_ptr->domain_index = 0;    /* First domain table entry */
    memset(update_domain_msg_ptr->id, 0, DOMAIN_ID_LEN);
    update_domain_msg_ptr->subnet       = subnet_id;
    update_domain_msg_ptr->must_be_one  = 1;    /* This bit must be set     */
    update_domain_msg_ptr->node         = node_id;
    update_domain_msg_ptr->len          = 0;
    memset(update_domain_msg_ptr->key, 0xFF, AUTH_KEY_LEN);
}

/*
 ****************************************************************************
 * create_assign_selector_msg().  Create the network management message to
 * assign a network variable selector to a network variable on a node.  The
 * message is an update NV config table entry network management message.
 ****************************************************************************
 */

void create_assign_selector_msg(msg_tag tag)
{
    NM_update_nv_cnfg_request *update_nv_cnfg_msg_ptr;
```

```
    /* Initialize network interface message buffer header and message tag. */
    ni_msg_hdr_init(sizeof(NM_update_nv_cnfg_request));
    if (node_id == NET_MGMT_NODE_ID)
        msg_out.ni_hdr.q.q_cmd = niNETMGMT;
    msg_out.msg_hdr.exp.tag = tag;

    /* Create destination address (subnet/node format). */
    msg_out.addr.snd.sn.type      = SUBNET_NODE;
    msg_out.addr.snd.sn.domain    = 0;
    msg_out.addr.snd.sn.node      = node_id;
    msg_out.addr.snd.sn.rpt_timer = RPT_TIMER_AT;
    msg_out.addr.snd.sn.retry     = RETRY_AT;
    msg_out.addr.snd.sn.tx_timer  = TX_TIMER_AT;
    msg_out.addr.snd.sn.subnet    = subnet_id;

    /* Create the update an NV config table entry message. */
    update_nv_cnfg_msg_ptr = (NM_update_nv_cnfg_request *) &msg_out.data.exp;

    update_nv_cnfg_msg_ptr->code           = NM_update_nv_cnfg;
    update_nv_cnfg_msg_ptr->nv_index       = 0; /* First NV                  */
    update_nv_cnfg_msg_ptr->nv_selector_hi = NV_SELECTOR_HI;
    update_nv_cnfg_msg_ptr->nv_direction   = (node_class == sensor) ? 1 : 0;
                                             /* 1 for output, 0 for input*/
    update_nv_cnfg_msg_ptr->nv_priority    = 0;
    update_nv_cnfg_msg_ptr->nv_selector_lo = NV_SELECTOR_LO;
    update_nv_cnfg_msg_ptr->nv_addr_index  = 0; /* First address table entry*/
    update_nv_cnfg_msg_ptr->nv_auth        = 0;
    update_nv_cnfg_msg_ptr->nv_service     = ACKD;
    update_nv_cnfg_msg_ptr->nv_turnaround  = 0;
}

/*
 ****************************************************************************
 * create_nv_msg().  Create an application buffer for an explicitly
 * addressed network variable poll or update.  Assume a 1 byte NV value.
 * Also assume that host selection has been enabled for the network
 * interface.  This is the default for the SLTA.
 ****************************************************************************
 */
```

```
void create_nv_msg(msg_tag tag)
{
    /* Initialize network interface message buffer header and message tag.  */
    /* The header is initialized assuming a 1 byte network variable value.  */
    /* Change the number 3 for different NV data sizes.                     */
    ni_msg_hdr_init(3);
    if (node_id == NET_MGMT_NODE_ID)
        msg_out.ni_hdr.q.q_cmd = niNETMGMT;
    msg_out.msg_hdr.exp.tag = tag;

    /* Create destination address (subnet/node format). */
    msg_out.addr.snd.sn.type      = SUBNET_NODE;
    msg_out.addr.snd.sn.domain    = 0;
    msg_out.addr.snd.sn.node      = node_id;
    msg_out.addr.snd.sn.rpt_timer = RPT_TIMER_AP;
    msg_out.addr.snd.sn.retry     = RETRY_AP;
    msg_out.addr.snd.sn.tx_timer  = TX_TIMER_AP;
    msg_out.addr.snd.sn.subnet    = subnet_id;

    /* Create the NV update or poll message. */
    msg_out.data.unv.NV_selector_hi = NV_SELECTOR_HI;
    msg_out.data.unv.direction      = (subnet_id == SENSOR_SUBNET) ? 1 : 0;
                                    /* Set to 1 for output NV, 0 for input  */
    msg_out.data.unv.must_be_one    = 1;
    msg_out.data.unv.NV_selector_lo = NV_SELECTOR_LO;
}

/*
 ****************************************************************************
 * create_query_status_msg().  Create the network management message to
 * query status from a node.
 ****************************************************************************
 */

void create_query_status_msg(msg_tag tag)
{
    /* Initialize network interface message buffer header and message tag. */
    ni_msg_hdr_init(1);
    if (node_id == NET_MGMT_NODE_ID)
        msg_out.ni_hdr.q.q_cmd = niNETMGMT;
    msg_out.msg_hdr.exp.tag = tag;
```